OIFIG NA BPAITINNÍ + CILL CHAINNIGH +

Patents Office
Government Buildings
Hebron Road
Kilkenny

JC882 U.S. PTO
09/735606
12/14/00

I HEREBY CERTIFY that annexed hereto is a true copy of documents filed in connection with the following patent application:

Application No.	S980713
Date of filing	31 August 1998
Applicant	TELLABS RESEARCH LIMITED, An Irish Company of Shannon Industrial Estate, Shannon, County Clare, Ireland.

CERTIFIED COPY OF PRIORITY DOCUMENT

Dated this 14th day of June 2000.

Gerard Barrett

An ~~officer~~ authorised by the
Controller of Patents, Designs and Trademarks.

FORM NO. 1

REQUEST FOR THE GRANT OF A PATENT

PATENTS ACT, 1992

The Applicant(s) named herein hereby request(s)

_____ the grant of a patent under Part II of the Act

X the grant of a short-term patent under Part III of the Act

on the basis of the information furnished hereunder.

1. Applicant(s)

Name

TELLABS RESEARCH LIMITED

Address

Shannon Industrial Estate
Shannon
County Clare
Ireland

Description/Nationality

An Irish Company

2. Title of Invention

"A Telecommunications System Controller Framework"

3. Declaration of Priority on basis of previously filed application(s) for same invention (Sections 25 & 26)

Previous filing date	Country in or for which filed	Filing No.
17/06/1998	Ireland	980474

4. Identification of Inventor(s)
Name(s) of person(s) believed by Applicants(s) to be the inventor(s)

Gerard Hartnett

Address
c/o Tellabs Research Limited
Shannon Industrial Estate
Shannon,
County Clare
Limerick

5. Statement of right to be granted a patent (Section 17(2) (b)

By virtue of an Assignment dated May 1, 1998

6. Items accompanying this Request – tick as appropriate

(i)	X	prescribed filing fee (£50.00)
(ii)	X	specification containing a description and claims
		specification containing a description only
	X	Drawings referred to in description or claims
(iii)	X	An abstract
(iv)		Copy of previous application (s) whose priority is claimed
(v)		Translation of previous application whose priority is claimed
(vi)	X	Authorisation of Agent (this may be given at 8 below if this Request is signed by the Applicant (s)

7. Divisional Application (s)

The following information is applicable to the present application which is made under Section 24 –

Earlier Application No: ....

Filing Date: ....

8. Agent

The following is authorised to act as agent in all proceedings connected with the obtaining of a patent to which this request relates and in relation to any patent granted -

Name	Address
John A. O'Brien & Associates	The address recorded for the time being in the Register of Patent Agents, and currently Third Floor, Duncairn House, 14 Carysfort Avenue, Blackrock, Co. Dublin, Ireland.

9. Address for Service (if different from that at 8)

As above

Signed [illegible] John A. O'Brien & Associates

Date August 31, 1998

"A Telecommunications System Controller Framework"

The invention relates to a telecommunications system controller framework.

Heretofore, the approach for telecommunications system controllers has been to link the software very closely with the hardware resources and provide much dedicated functionality. This approach can provide a fast response time for real time telecommunications control, however, it lacks flexibility and modifications are difficult to make.

Therefore, one object of the invention is to provide a framework which operates efficiently in real time, and which may be easily modified to cater for changing telecommunications resources and volume requirements.

According to the invention, there is provided a telecommunications system controller framework comprising:-

application domain base objects in a base level; and

meta data for the base objects in a meta level.

Preferably, the meta data comprises meta objects, and there is preferably one meta object associated with each base object class. For example, there may be a base object associated with each slot in a telecommunications exchange rack, and one meta object associated with the slot class.

Preferably, the meta level operates in reaction to the base objects. Thus, the base objects perform the application processing and either make requests to the meta objects or transfer certain information to the meta objects so that they can react.

In one embodiment the meta data includes base object containment information. Preferably, the meta data comprises means for using the containment information for interrogating the base object containment hierarchy to locate a required object in response to a request from a requesting base object. By storing the containment information, the meta level can interrogate the base level to locate required objects.

In one embodiment, the meta data includes persistence data. This allows the base objects to be free from the need to perform any data persistence operations, thus helping to provide a fast response time, and also consistency in the manner in which persistence data is dealt with.

In one embodiment, the meta objects comprise means for performing persistence data operations transparently to the base objects.

In another embodiment, the meta data includes real resource information for attributes of the base objects. Preferably, the meta objects comprise means for verifying base object proposals to update real resource attributes, and for performing the updates if approved. In one embodiment, the meta data objects comprise means for publishing real resource update events on a channel. The base objects are divorced from the real resources, thus allowing flexibility as the base objects may be modified to cater for different functionality without the need to deal with the manner in which they interface with the real resources. Location on a channel is an effective way of disseminating real resource update information.

In one embodiment, the meta data includes condition and alarm information.

Preferably, the meta objects include controller backup information and means for updating a backup controller.

The invention will be more clearly understood from the following description of some embodiments thereof, given by way of example only with reference to the accompanying drawings in which:-

Fig. 1 is a diagrammatic representation of a framework; and

Figs. 2 to 15 inclusive are diagrams indicating interaction of layers in the framework.

Referring now to Fig. 1, a framework 1 for a telecommunications system controller is illustrated. In this embodiment, the telecommunications system is for handling ATM cells. The framework 1 comprises an application domain base level 2 having base objects 3. It also comprises a meta level 4 comprising meta objects 5. In the example illustrated, the base objects 3 are for telecommunications system card slots. This is a class of base object and there is one meta object 5 for each class.

Briefly, the base objects 3 transmit various requests and signal actions to the relevant meta object 5. The meta object 5 accesses the database 6 and carries out various actions relating to base object containment, persistence data, real resource update, and controller backup updating. The meta level 4 interfaces with real resources 10.

Referring now to Fig. 2, a general scenario illustrating operation of the framework 1 is illustrated. In this illustration, a slot object 3 accesses a corresponding meta object 5, which in turn accesses a resource attribute adapter 11. In general, the base level defines the application logic, while the meta level performs various actions which can be more efficiently performed by the meta objects. The meta level describes a number of different aspects of the base level software including containment information, information on which attributes are persistently stored, real resource information, and backup information.

Thus, complex processes such as downloading all of the information in an object hardware can be performed by the meta level 4.

Referring to Fig. 3, an example is illustrated whereby an attribute is set using a base level function. Application code calls a setLineCoding method on a SlotDo base object 3. This informs the meta level, calling the setFourBaseLevel function on the meta attribute. The meta attribute verifies the new value can indeed be changed from its point of view. If, for example, the attribute was representing a value on the line card, the real resource aspect of the meta object would have provided a verification strategy object to the meta attribute. In this case the verification strategy object would try setting the attribute on the line card.

Referring now to Fig. 4, an example is illustrated whereby a parent object creates a contained object. When doing this, it informs the meta level of the new object using notifyCreate. There are three phases as follows:-

(a) construction, in which the object is allocated,

(b) initialisation, in which it is fully set up, and

(c) notification, in which the meta object is notified.

These three phases are controlled by the parent object.

As stated above, there is one meta object per class of domain object. Meta objects are containers for other objects which describe the domain object class and contain attributes, actions, an other information such as containment and persistence information. The meta objects also contain event channels which are informed when various events take place. For example, subscribers can learn of instance creation and deletion using these channels. In the example of Fig. 3, a persistence object acts as a subscriber receiving notifications of instance creations.

In more detail, the base objects represent the problem domain and include card, slot and ATM interface objects. They are implemented as standard C++ objects and have get/set methods for attributes. The method may use methods on other base objects and they may create their contained objects and may contain state, state machines and support alarm conditions. The base objects invoke services on the meta level. In an example described above, where an attribute is being set on the base level the base level invokes a method on the meta level informing it of attribute value changes.

The attributes are contained in the meta objects 5. Attributes can be get/set from the meta level as well as from the base level. This is used when attribute values are coming from the meta level as well as from the base level, such as from persistent storage. Referring to Fig. 5, a mechanism for getting and setting attributes is described. The attribute calls the base level functions for setting and getting the value.

Meta attributes are declared using a hierarchy of abstract classes to avoid including much code in header files. Referring to Fig. 6, a mechanism for getting an attribute from the meta level is illustrated.

Meta actions support the domain concept of actions in the domain object. They are informed when the base level is running an "action". They can also run the action from the meta level.

Keys are simple classes which represent the information needed to uniquely identify an instance of a class. Keys are made of inheritance chains. If an object of the class can be contained in objects of another class then the set of that class will inherit from its container's key class. In this example, Slot is contained in Shelf. Keys can be used to find an instance of a base object. This behaviour is supported from the meta level containment information, as illustrated in Fig. 7. The opposite scenario can also happen, namely an instance can set a key value. To do this the instance sets the part of the key it understands. It needs its parent object to set the rest of the key, and so on up the tree.

The containment information of the meta level provides the following functionality:

- finding a domain object using a stream representation of its identity.

- creating a new domain object identified by a string representation of its identity.

- creating an iterator for instances of the domain class.

- finding an object based on a key representation of its identity.

The "contain info" needs to know information such as the string representation of the class name, and this is obtained from the meta object. It also needs the containment information from its container classes to perform some of its functionality. Accordingly, instances of containment information objects tend to be changed together in a way that reflects the containment tree.

There are a number of different concrete containment information classes which implement the generic behaviour specified by the abstract ContainInfoFor. These are:

- SingleParentcontainInfoFor – a contain info for a class of objects which is directly contained by a singleton. The objects are not unique within their container and so need an identifier to specify them uniquely within a container.

- UniqueContainInfoFor - a contain info for a class of objects which are unique within their container (e.g. there can be only one Card in a Slot).

- ChildContainInfoFor – stores containment meta-level information for objects which can be contained in other objects. The objects are not unique within their container and so need an identifier to specify them uniquely within a container.

Referring to Fig. 9, an example problem domain is illustrated in which there is a singleton representing the whole system, shelves are contained in the system, slots and fans are contained in a shelf and cards are uniquely contained in a slot.

Referring to Fig. 10, ChainedContainInfo classes for the above problem domain containment tree are illustrated. The ShelfMeta contains the SingleParentContainInfoFor. It also contains ContainInfoAdapers for its two types of contained objects.

The meta level classes for containment need to use facilities implemented by the base level. These facilities are defined in the following two abstract classes:-

- ContainerOf – specifies the class is a container of another type of object. Means findObject, findFirstObject and findNextObject, all taking an identifier, must be implemented.
- ContainerOfUnique – specifies the class is a container of a unique object. Means findObject taking no identifier must be implemented.

Referring to Fig. 11, the class declarations for the domain objects Shelf and Slot are illustrated.

Some of the attributes of the base objects need to be persistently stored and this is handled by a meta object. The meta object contains an "AbstractPersistObject" adapter in the same way. The AbstractPersistObject provides interfaces whereby it can load all objects of a class from persistence and can subscribe to object creations and deletions. It also contains AbstractPersistAttr objects which are created for each attribute which is persistent.

There are two different concrete types of AbstractPersistObject. The PersistStaticObjectAdapterFor is for domain objects which are expected to exist before the loadObjectsFromPsl function is called. There are a fixed number of them in the system and they merely need to be initialised.

The loadObjectFromPsl goes through each entry in the persistence storage layer, finds the existing object that refers to the entry, then uses the AbstractPersistAttr objects to extract the information from the raw persistence storage and set the appropriate attributes.

The other concrete AbstractPersistObject is the PersistDymanicObjectAdapterFor. This concrete class is used where the instances are not expected to pre-exist before the loadObjectsFromPsl call and the persist adapter will need to cause them to be created. The container of the object will do the actual creating and as such will need to implement the makeObject function.

Referring to Fig. 12, the declarations for setting an attribute are illustrated. The AbstractPersistAttr subscribes to changes in the meta attribute and persistently stores the new values when changes do happen. The mechanism for loading an object from persistence storage is illustrated in Fig. 13.

In the example illustrated, the contain info is passed to the whole stream. It finds the container and calls makeObject on. If the whole operation is successful a new object is returned. This is then initialised using the storage from the persistence layer. Finally, the notifyCreate is called on the meta level.

Three phases of construction used to form the persistence layer are shown in Fig. 14. The PersistDymanicObjectAdapterFor uses the makeObject to create the object. It uses setObjectFromStorage to initialise it. It then informs the meta level using the notifyCreate.

The meta level is also used for tracking and updating real resources. Typically, there are proxy objects for interacting with objects on the real resource. There are two scenarios:-

(a) When the attribute is set the real resource must try to set the value first. This results in communicating with a middleware proxy. If this works, the rest of the process of setting an attribute is run.

(b) If the real resource needs to be configured from the controller, all of the real resource attributes need to be downloaded to the hardware. An action means calling a proxy function on the real resource. The proxy function can be called and return-checked in the base object function. If an attribute is read-only on the hardware and it can change autonomously, the Getfunction in the domain object can call the proxy directly to get the value from the hardware.

A mechanism for setting an attribute on a real resource is illustrated in Fig. 15.

The meta level also provides an agent interface to the base objects for the agent sub-system. Some attributes may not be visible on the agent interface or they may be more restricted in their scope i.e. read-only instead of read/write.

Two other aspects of base objects and their representation as meta objects are polymorphism and text interfaces. Polymorphism is where there is a base class domain object from which derived classed are to be inherited. The derived classes want to add new attributes and actions and so will have extra meta information as well as the information from the base class.

The text interface provides facilities for representing objects and attributes as text strings. It is used by the CLI software.

The meta-level architecture may also support domain object versioning.

It will appreciated that the invention allows application code and logic to be independent of application level services. This means that changes to horizontal application services can be made without affecting the general application logic, avoiding sweeping changes to the software system. These advantages have been achieved with very simple and well defined interfaces between the base and the meta level.

New functionality can be added to the system without affecting the application code because must of the persistence, real resource interface, text interface, and auditing functionality is written in terms of the meta level.

The invention also achieves a high level of reuse, and qualification of the system is simplified.

The invention is not limited to the embodiments described, but may varied in construction and detail within the scope of the claims.

Claims

1. A telecommunications system controller framework comprising:-

   application domain base objects in a base level; and

   meta data for the base objects in a meta level.

2. A framework as claimed in claim 1, wherein the meta data comprises meta objects, and wherein there is one meta object associated with each base object class, and wherein the meta level operates in reaction to the base objects and, wherein the meta data includes base object containment information, and wherein the meta data comprises means for using the containment information for interrogating the base object containment hierarchy to locate a required object in response to a request from a requesting base object.

3. A framework as claimed in claims 1 or 2, wherein the meta data includes persistence data, and wherein the meta objects comprise means for performing persistence data operations transparently to the base objects, and wherein the meta data includes real resource information for attributes of the base objects, and wherein the meta objects comprise means for verifying base object proposals to update real resource attributes, and for performing the updates if approved, and wherein the meta data objects comprise means for publishing real resource update events on a channel.

4. A framework as claimed in any preceding claim, wherein the meta data includes condition and alarm information, and wherein the meta objects include controller backup information and means for updating a backup controller.

5. A framework substantially as described with reference to the drawings

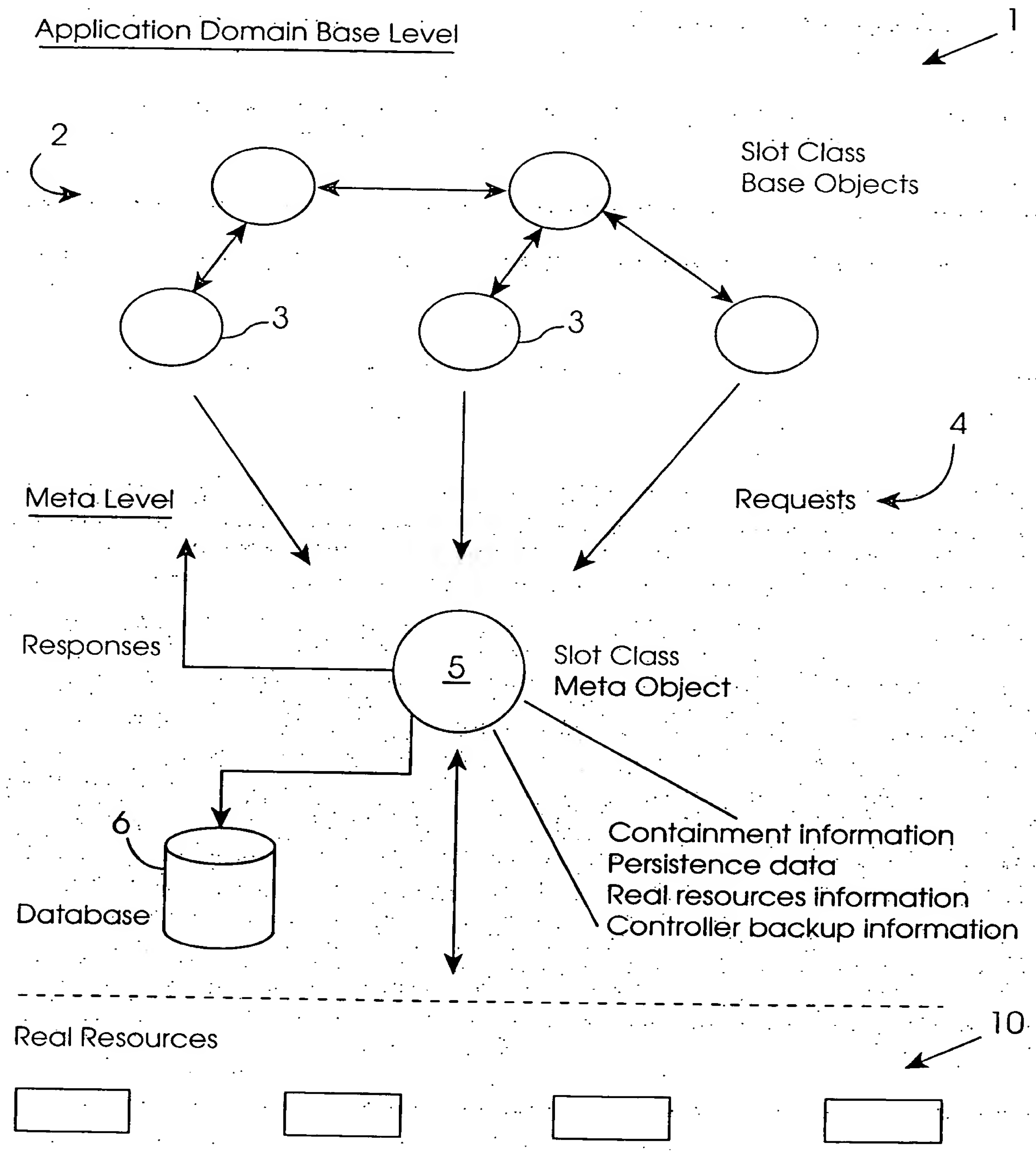
Application Domain Base Level
1
2
Slot Class
Base Objects
3
3
4
Requests
Meta Level
Responses
5
Slot Class
Meta Object
6
Database
Containment information
Persistence data
Real resources information
Controller backup information
Real Resources
10

Fig. 1

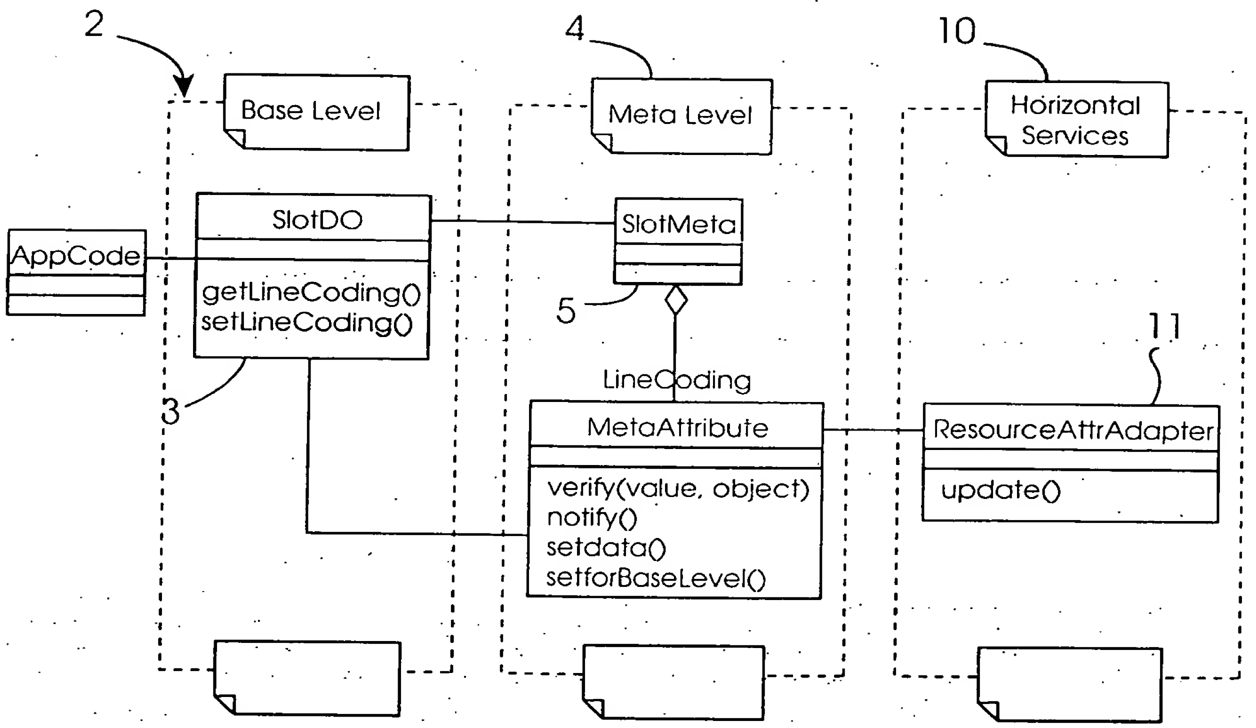
2
4
10
Base Level
Meta Level
Horizontal Services
SlotDO
getLineCoding()
setLineCoding()
AppCode
SlotMeta
5
3
LineCoding
MetaAttribute
verify(value, object)
notify()
setdata()
setforBaseLevel()
11
ResourceAttrAdapter
update()

Fig. 2

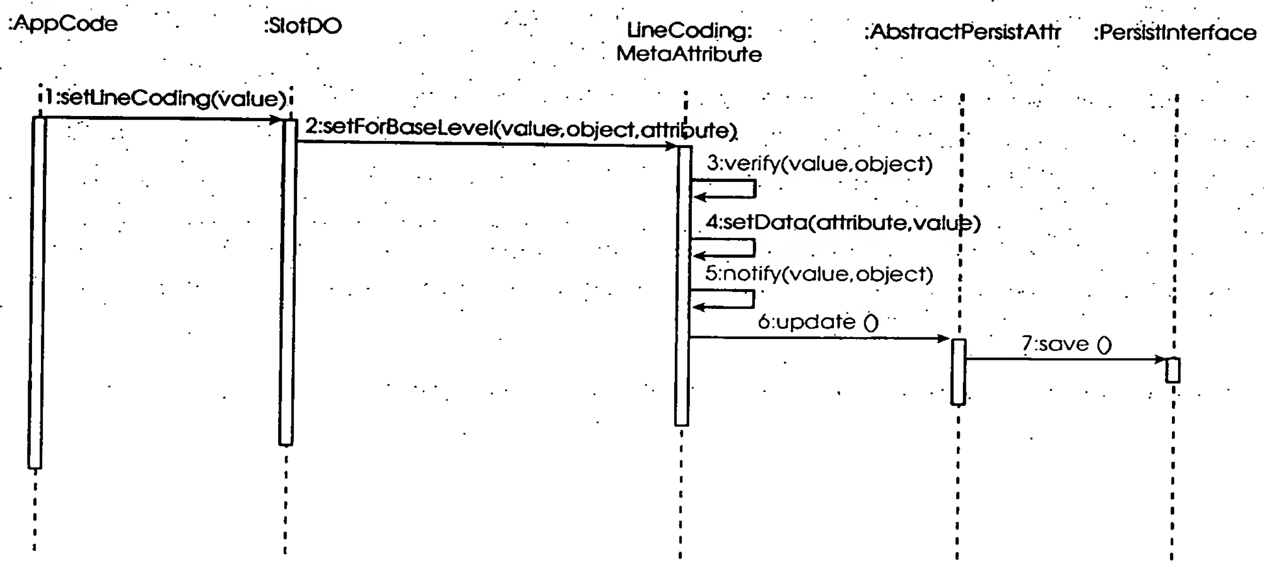
:AppCode
:SlotDO
LineCoding: MetaAttribute
:AbstractPersistAttr
:PersistInterface
1:setLineCoding(value)
2:setForBaseLevel(value,object,attribute)
3:verify(value,object)
4:setData(attribute,value)
5:notify(value,object)
6:update ()
7:save ()

Fig. 3

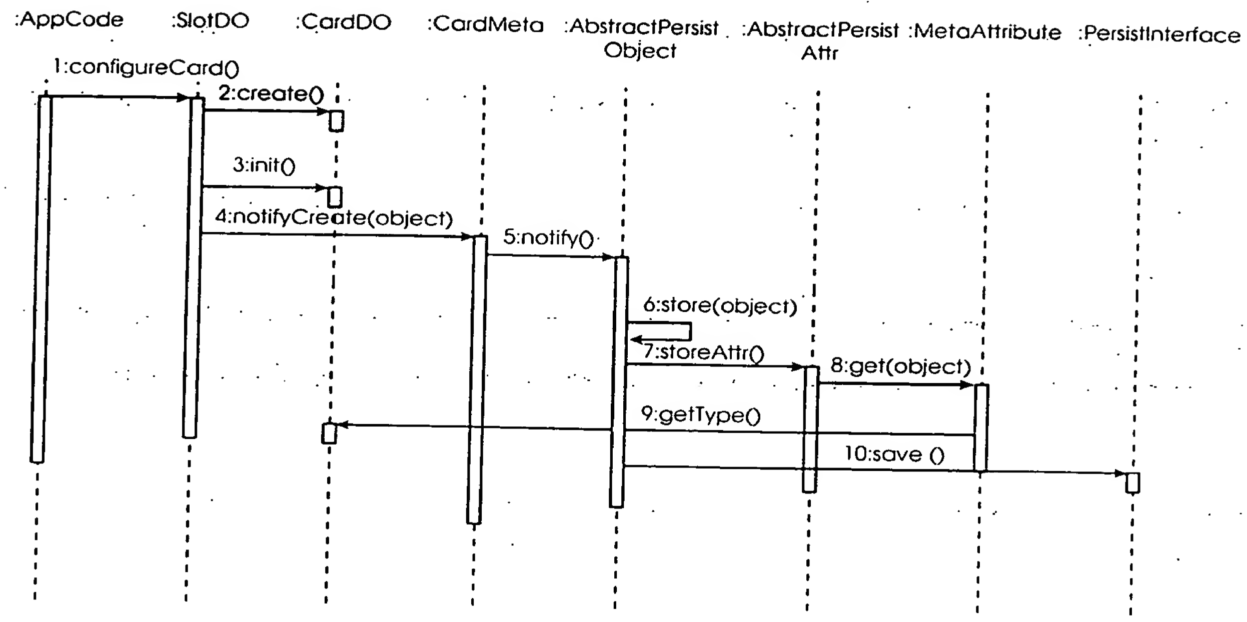

diagram: CreatingAnObject

Fig. 4

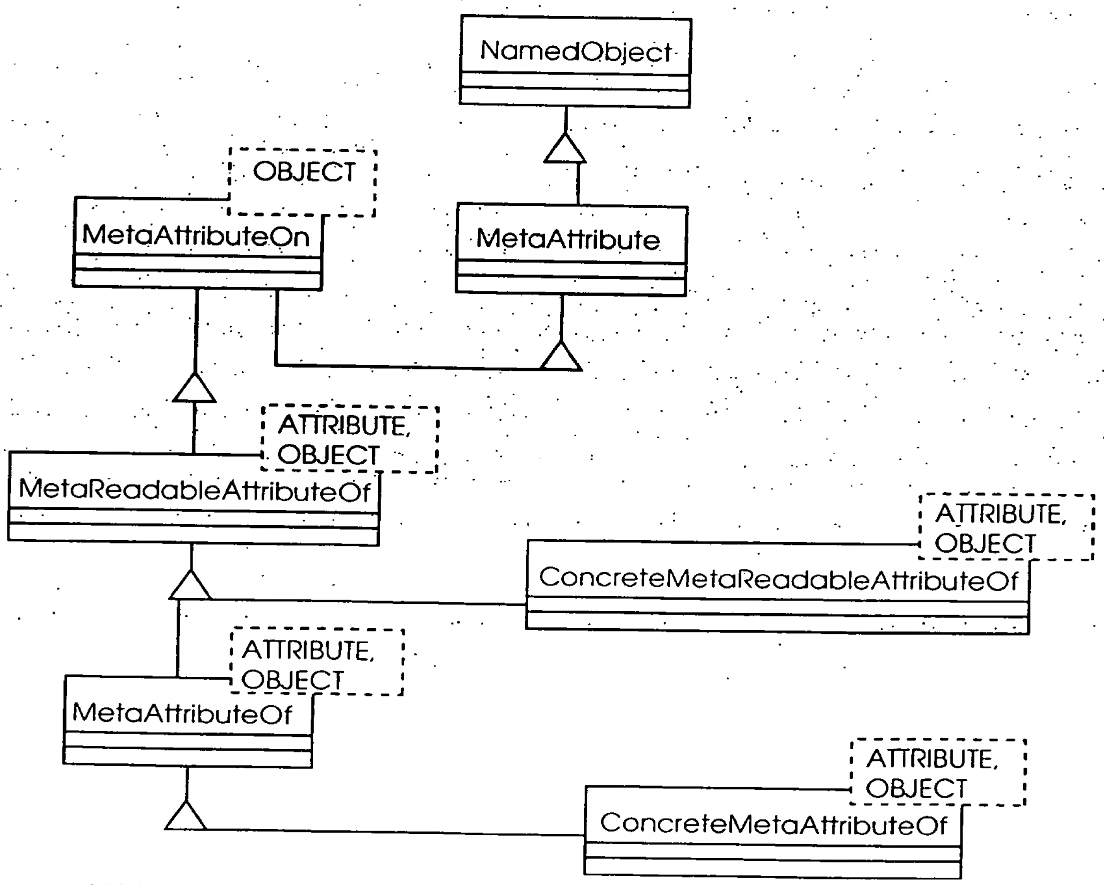

diagram: AttributeClasses

Fig. 5

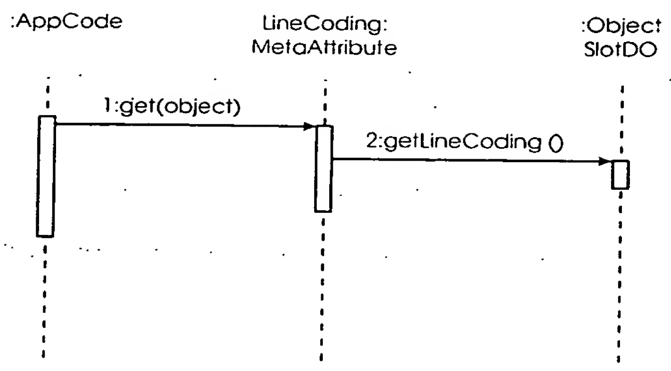

diagram:GettingAnAttributeFromMeta

Fig. 6

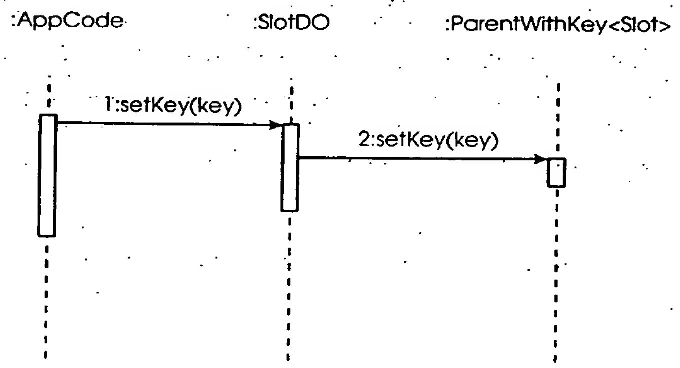

Fig. 7

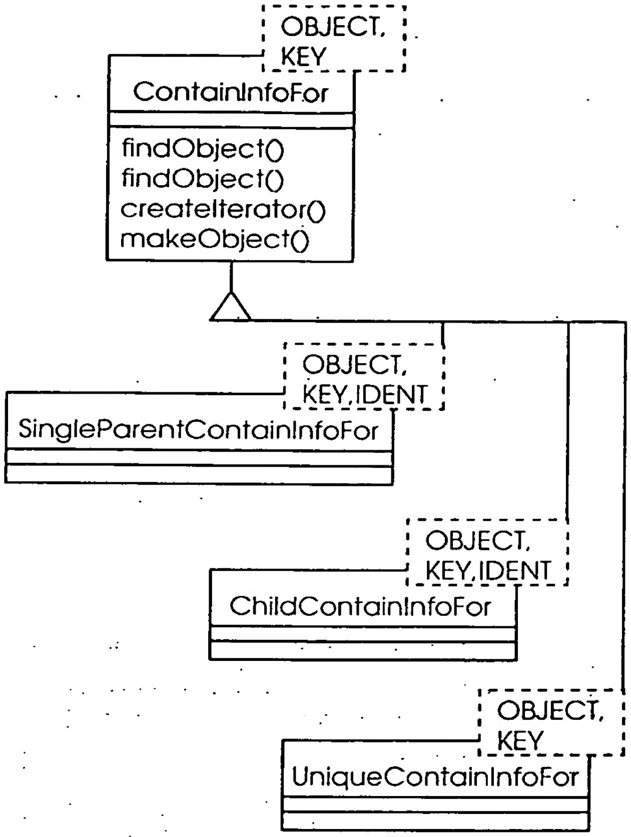
OBJECT, KEY
ContainInfoFor
findObject()
findObject()
createIterator()
makeObject()
OBJECT, KEY,IDENT
SingleParentContainInfoFor
OBJECT, KEY,IDENT
ChildContainInfoFor
OBJECT, KEY
UniqueContainInfoFor

Fig. 8

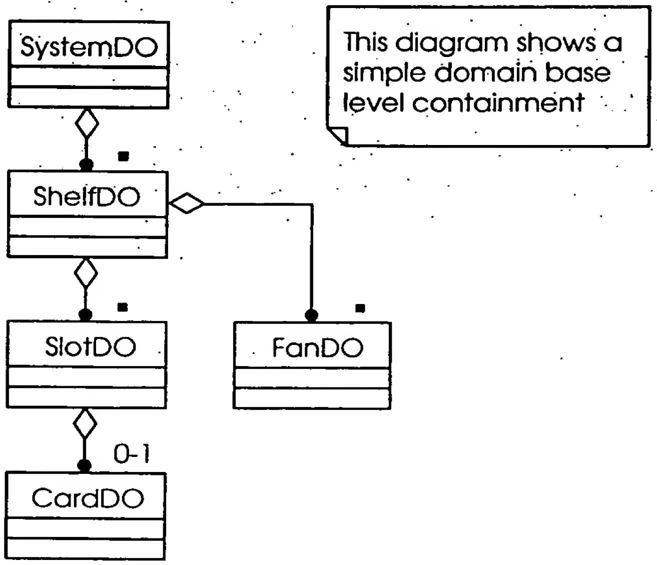
SystemDO
ShelfDO
SlotDO
FanDO
CardDO
0-1
This diagram shows a simple domain base level containment

Fig. 9

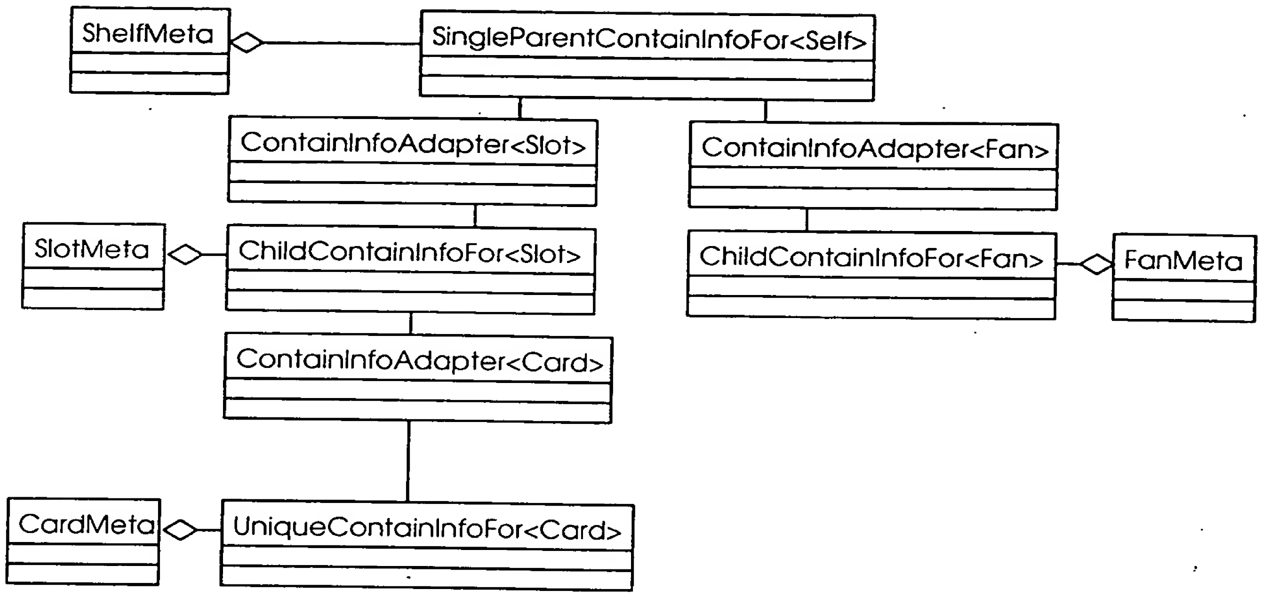

Fig. 10

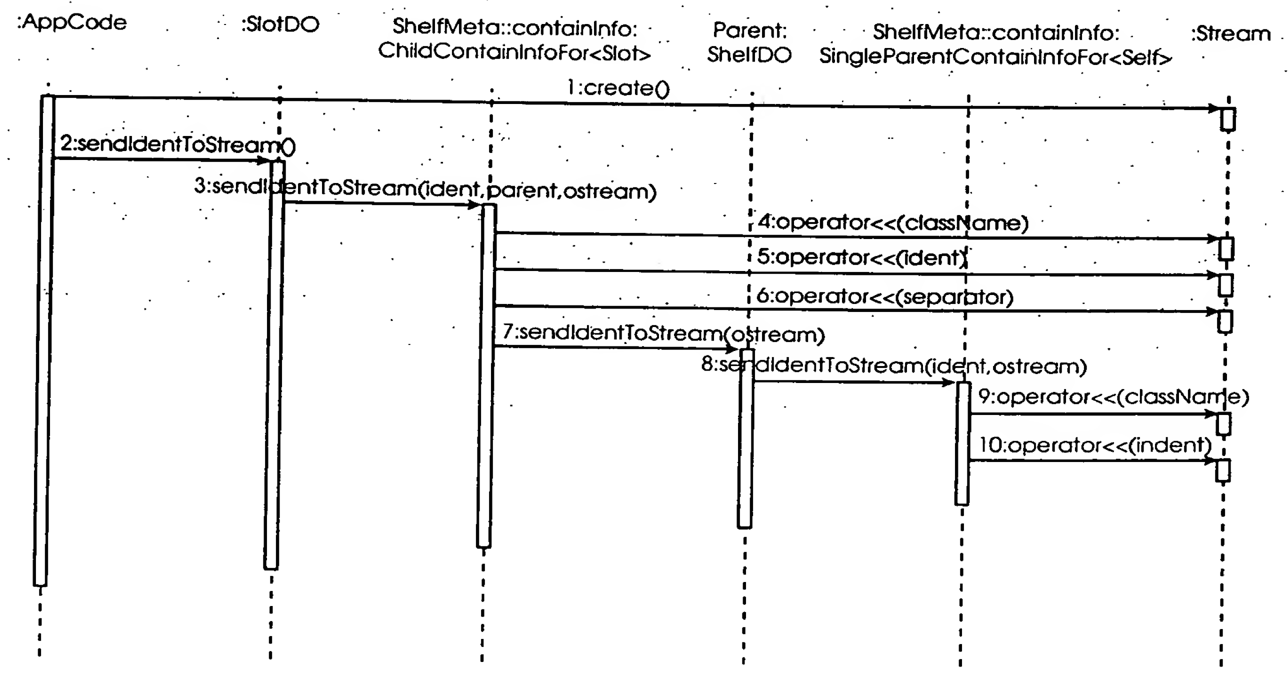
:AppCode
:SlotDO
ShelfMeta::containInfo: ChildContainInfoFor<Slot>
Parent: ShelfDO
ShelfMeta::containInfo: SingleParentContainInfoFor<Self>
:Stream
1:create()
2:sendIdentToStream()
3:sendIdentToStream(ident,parent,ostream)
4:operator<<(className)
5:operator<<(ident)
6:operator<<(separator)
7:sendIdentToStream(ostream)
8:sendIdentToStream(ident,ostream)
9:operator<<(className)
10:operator<<(indent)

diagram:SendingAnObjectIdToAStream

Fig. 11

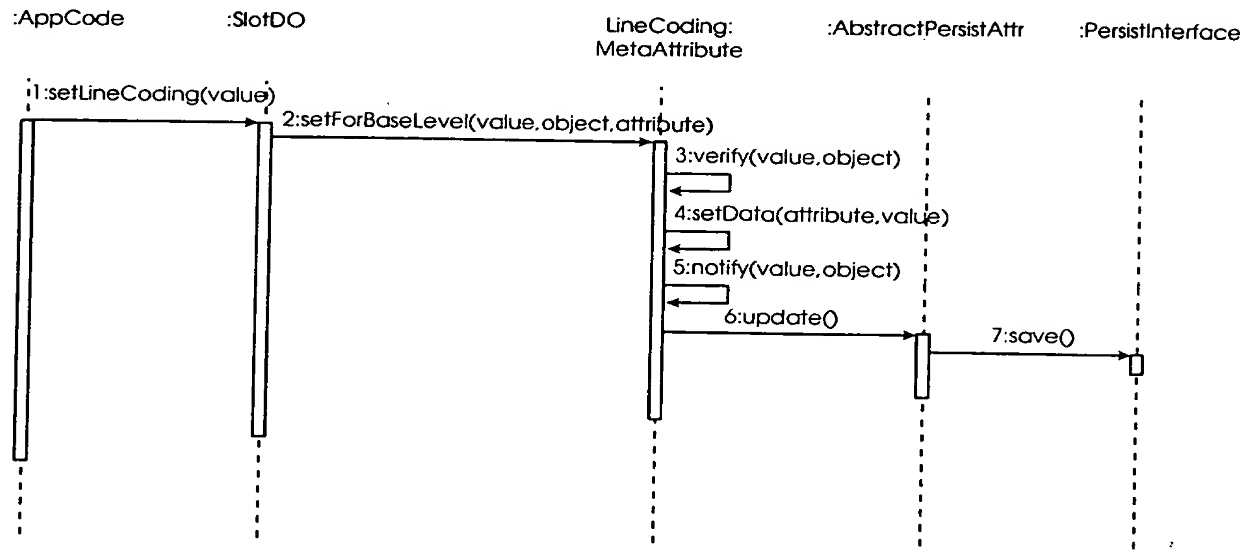
:AppCode
:SlotDO
LineCoding:
MetaAttribute
:AbstractPersistAttr
:PersistInterface
1:setLineCoding(value)
2:setForBaseLevel(value,object,attribute)
3:verify(value,object)
4:setData(attribute,value)
5:notify(value,object)
6:update()
7:save()
diagram: SettingAnAttribute

Fig. 12

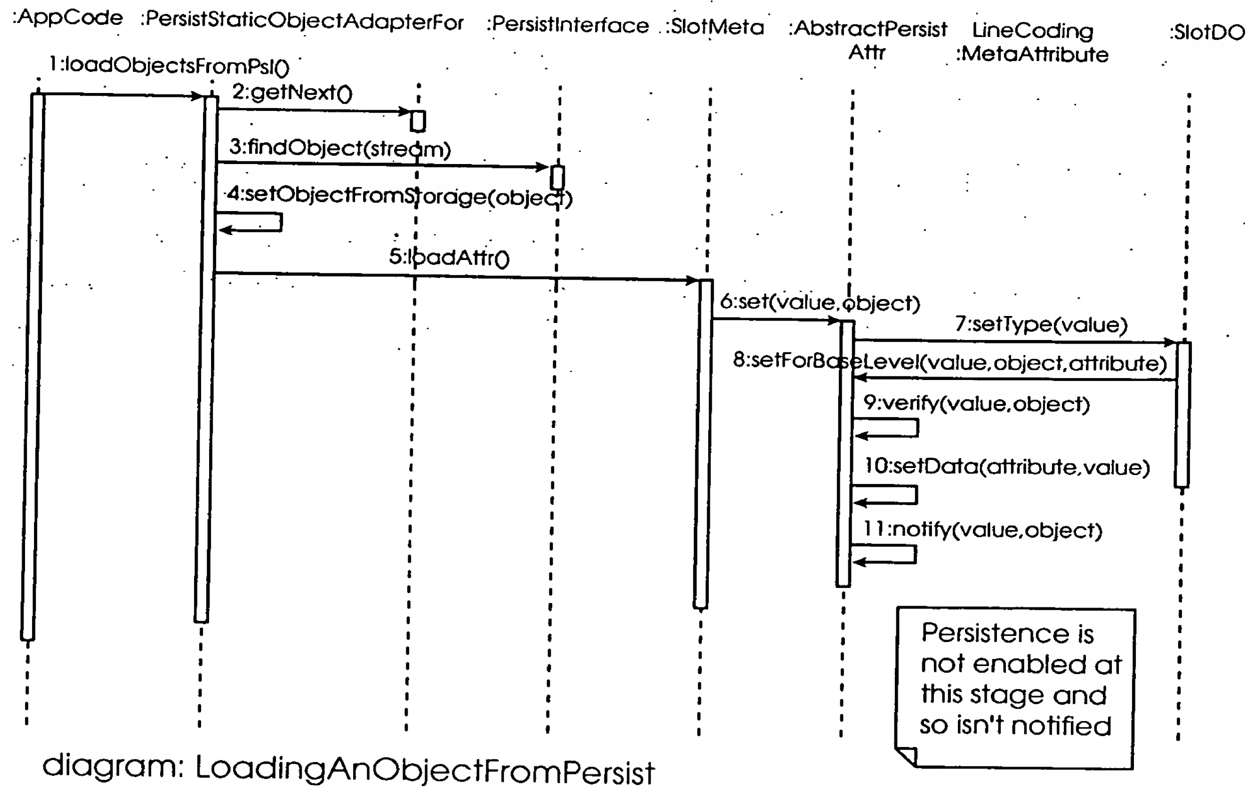
:AppCode
:PersistStaticObjectAdapterFor
:PersistInterface
:SlotMeta
:AbstractPersist
Attr
LineCoding
:MetaAttribute
:SlotDO
1:loadObjectsFromPsI()
2:getNext()
3:findObject(stream)
4:setObjectFromStorage(object)
5:loadAttr()
6:set(value,object)
7:setType(value)
8:setForBaseLevel(value,object,attribute)
9:verify(value,object)
10:setData(attribute,value)
11:notify(value,object)
Persistence is
not enabled at
this stage and
so isn't notified
diagram: LoadingAnObjectFromPersist

Fig. 13

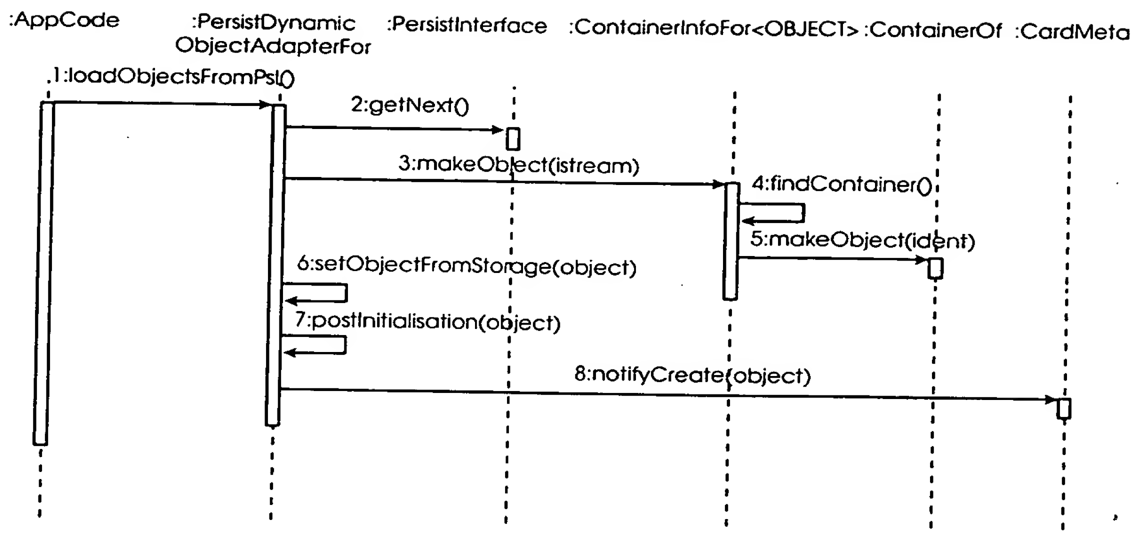

diagram: CreatingObjectFromDynamicPersist

Fig. 14

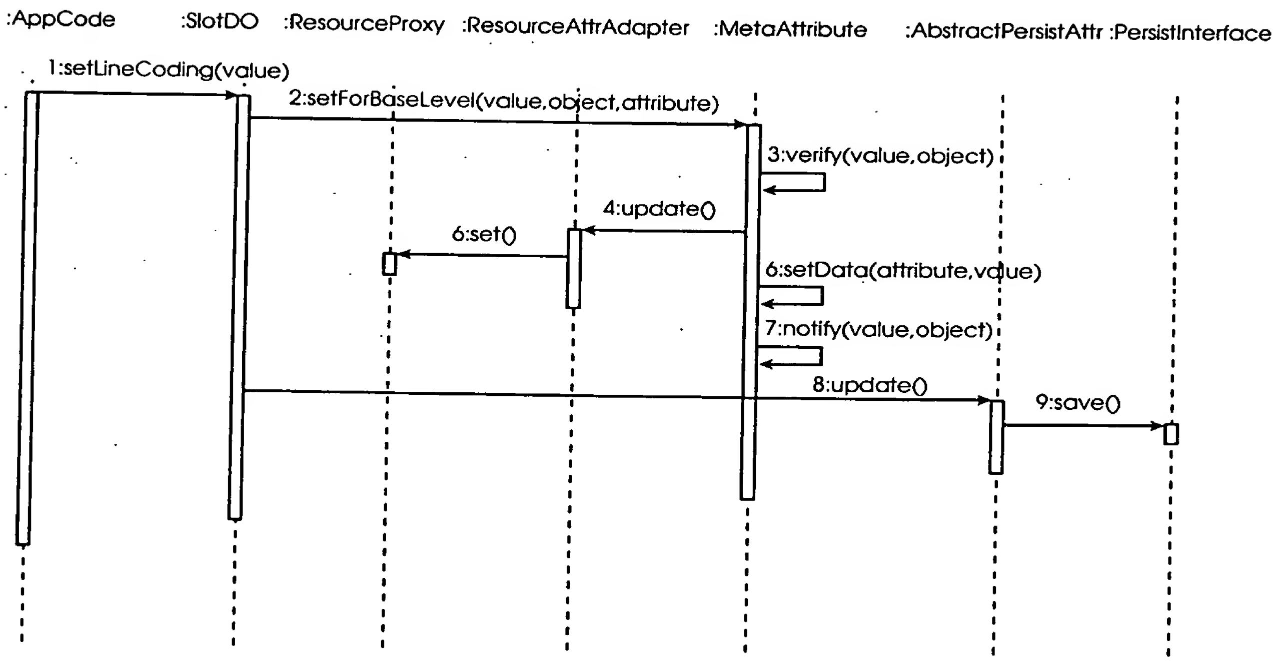

diagram: SettingAnAttributeOnResource

Fig. 15